XGBoost的主要优点：

1. 简单易用。相对其他机器学习库，用户可以轻松使用XGBoost并获得相当不错的效果。
2. 高效可扩展。在处理大规模数据集时速度快效果好，对内存等硬件资源要求不高。
3. 鲁棒性强。相对于深度学习模型不需要精细调参便能取得接近的效果。
4. XGBoost内部实现提升树模型，可以自动处理缺失值。

XGBoost的主要缺点：

1. 相对于深度学习模型无法对时空位置建模，不能很好地捕获图像、语音、文本等高维数据。
2. 在拥有海量训练数据，并能找到合适的深度学习模型时，深度学习的精度可以遥遥领先XGBoost。

这里我们采用-1将缺失值进行填补，还有其他例如“中位数填补、平均数填补”的缺失值处理方法有兴趣的同学也可以尝试。

```
data = data.fillna(-1)
```

我一般使用平均数填补

一些相关pd笔记

```
age_mean = data.Age.mean()
# age_mean = data['Age'].mean()

data.Age[data['Age'].isnull()] = age_mean
# data['Age'] == data.Age 二者是等效的
```

哈哈哈，真的好几次犯过这样的错误呢
需要注意的是，第一种方法看似像是一种列表形式，但是如果像是如下方式书写代码会报错

```
df['goods','quantity']    #会报错，不要妄想通过这种方式获取多列。
```

上方的形式会引发一个KeyError的错误。如果想通过第一种形式获取多列，正确的形式如下

```
df[['goods','quantity']]
```

上面代码的意思是选取DataFrame中，goods列的值为’ eggplant ‘的所有行。
DataFrame中也支持多条件筛选，与Python的判断语句相同，and、or、not 和 xor分别代表 和、或、非、抑或。也可以用符号代替 &、|、~、^。每个判断条件要用圆括号括起来，否则会报错

```
df[(df.goods =='eggplant') & (df.quantity == 12)]
```

选取df中 goods列值为 eggplant且 对应的 quantity值为12的所有列。

在筛选数据时候也可以使用loc和iloc函数实现子集的选取，意思并不是说上方的筛选语句不对，亲测同样很好用，而且代码简单了点。loc和iloc进行数据筛选的格式如下：

```
df = df.loc[df.goods=='eggplant', :]
```

.loc[]相当于是把想要筛选的列通过 ':'全部选出来。loc方法同样支持多条件筛选。

```
df = df.loc[(df.goods =='eggplant') & (df.quantity == 12), :]
```

loc形式的多条件筛选，条件与条件之间同样要使用括号分割开。这两行loc方法筛选的结果与之前没用loc方法写的代码效果是一样的，格式规范而言肯定是有loc的更容易让人理解。但是如果只是给自己写脚本完全可以用前一种，必要时候做个注释即可。

条件筛选的另一种形式，就是选取一个DataFrame中满足条件的行的某几列（并不是全部列）。此时只需将上面的代码做一小点改变即可。

```
df = df.loc[(df.goods =='eggplant') & (df.quantity == 12), ['goods', 'price']]
1
```

此时亲测使用loc方式会比较好一些，因为之前loc形式后方的 ‘:’ 代表的是满足条件的全部行的所有列的数据，一个冒号囊括了所有列。此时要是指定其中的某几列，通过list的形式，list的元素是DataFrame的列名，元素类型是字符串。上方的代码所表达的是满足条件的所有行的 goods 和 price 两列。
.loc还是很有用的，筛选的时候经常使用

增删改查用的比较多比较熟悉了，就不记录了

数据类型转化

dataframe中的astype()方法是对数据类型进行转换的函数。以一个例子进行说明

```
df = pd.DataFrame({'id': range(4), 'age': ['13', '34', '23'], 'weight': ['45.7', '60.9', '55.5']})
df.dtypes
```

dtype的结果会显示出，id列的数据类型是 int， 而age和weight列的数据类型都是object。现在将age转化为 int类型，weight转化为float类型。此时astype函数的作用就有所体现。

```
df.astype({'age':'int', 'weight':'float'})
```

通过字典的形式传入，将两列转换成了想要的数据类型。

数据排序

dataframe中索引排序用到的是sort_index方法，相似的列数值排序运用的是sort_values()方法。

```
df.sort_values( by=['age'], ascending=False, inplace=True）#只通过age进行排序；升序；如有缺失值放在最后。
df.sort_values(by=['age', 'weight'], ascending=True, inplace=True, na_position='first') #先通过age排序，在其基础上通过weight排序；降序，缺失值放在最前方。
```

数据的排序在实际问题中很有用，比如找出聚类中得分最高的几个项等等。

数据去重

在dataframe中去除重复，使用drop_duplicates()方法。而只是检查dataframe中是否出现重复有的是duplicated方法，如果只是想检查某列中是否存在重复的值可以通过subset参数将列传入到duplicated()方法中。

```
df.duplicated()  #检查整个dataframe中是否出现重复
df.duplicated(subset='age')  #检查age这列是否出现重复数据
df.drop_duplicates()  #删除dataframe中的重复
```

其中，subset参数传入的可以是单一的列名；同时也可以是多个列名组成的list。

频率统计

dataframe中，统计某列不同的值出现的次数使用value_counts()方法。

df = pd.DataFrame({'id': range(4), 'name': ['Jack', 'Craig', 'Chuck', 'Jack'],'gender': ['M', 'M', 'M', 'F'] , 'age': ['13', '34', '23','4'], 'weight': ['45', '60', '55', '30']})

可以看到上面的dataframe中，name列有重名的，现在要统计不同的名字出现的次数。

df.name.value_counts()

上面的代码反悔的结果是在这个dataframe中，不同的名字出现的次数。

value_counts()方法的一个妙用是用于求占比，假如现在要求的是性别的占比。

df.gender.value_counts()/ sum(df.gender.value_counts)

这行代码返回的是不同性别人数除以总人数所得到的个性别的占比。但是value_counts()方法只能是单变量的计数。如果想设置一个条件后在进行计数，可以使用的是crosstab()方法。
crosstab()还真的是第一次见
pd.crosstab(index=df.name, columns=df.gender)

这行代码，记录的是不同名字出现的次数，但是将性别分开讨论。columns参数可以传入多个列，传入形式是list，元素是列名。注意：columns传入的列必须是离散型变量，用途是用于分类

Step5:对离散变量进行编码

由于XGBoost无法处理字符串类型的数据，我们需要一些方法讲字符串数据转化为数据。一种最简单的方法是把所有的相同类别的特征编码成同一个值，例如女=0，男=1，狗狗=2，所以最后编码的特征值是在[0,特征数量−1]之间的整数。除此之外，还有独热编码、求和编码、留一法编码等等方法可以获得更好的效果。

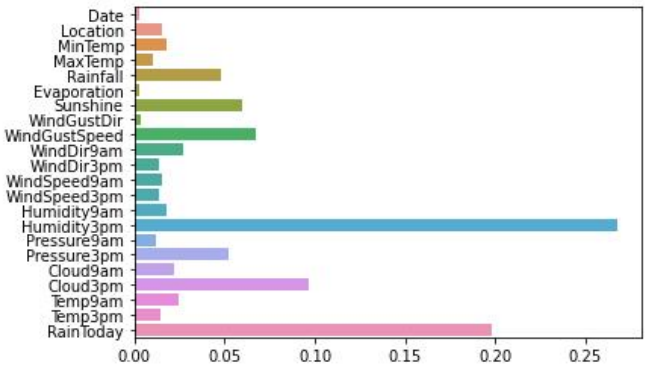
```
## 把所有的相同类别的特征编码为同一个值
def get_mapfunction(x):
```

```
    mapp = dict(zip(x.unique().tolist(),
         range(len(x.unique().tolist()))))
    def mapfunction(y):
        if y in mapp:
            return mapp[y]
        else:
            return -1
    return mapfunction
for i in category_features:
    data[i] = data[i].apply(get_mapfunction(data[i]))
```

onehotencode 应该就可以了

```
1  sns.barplot(y=data_features_part.columns, x=clf.feature_importances_)
```

```
[26]: <AxesSubplot:>
```



这个feature_importances_确实有用，特征工程一直都是预测分析的重要部分

除次之外，我们还可以使用XGBoost中的下列重要属性来评估特征的重要性。

- weight:是以特征用到的次数来评价
- gain:当利用特征做划分的时候的评价基尼指数
- cover:利用一个覆盖样本的指标二阶导数（具体原理不清楚有待探究）平均值来划分。
- total_gain:总基尼指数
- total_cover:总覆盖

```
from sklearn.metrics import accuracy_score
from xgboost import plot_importance
```

```
def estimate(model,data):

    #sns.barplot(data.columns,model.feature_importances_)
    ax1=plot_importance(model,importance_type="gain")
    ax1.set_title('gain')
    ax2=plot_importance(model, importance_type="weight")
    ax2.set_title('weight')
    ax3 = plot_importance(model, importance_type="cover")
    ax3.set_title('cover')
    plt.show()
def classes(data,label,test):
    model=XGBClassifier()
    model.fit(data,label)
    ans=model.predict(test)
    estimate(model, data)
    return ans


ans=classes(x_train,y_train,x_test)
pre=accuracy_score(y_test, ans)
print('acc=',accuracy_score(y_test,ans))
```

Step8: 通过调整参数获得更好的效果

XGBoost中包括但不限于下列对模型影响较大的参数：

1. learning_rate: 有时也叫作eta，系统默认值为0.3。每一步迭代的步长，很重要。太大了运行准确率不高，太小了运行速度慢。
2. subsample：系统默认为1。这个参数控制对于每棵树，随机采样的比例。减小这个参数的值，算法会更加保守，避免过拟合, 取值范围零到一。
3. colsample_bytree：系统默认值为1。我们一般设置成0.8左右。用来控制每棵随机采样的列数的占比(每一列是一个特征)。
4. max_depth： 系统默认值为6，我们常用3-10之间的数字。这个值为树的最大深度。这个值是用来控制过拟合的。max_depth越大，模型学习的更加具体。

网格调参法

```
## 从sklearn库中导入网格调参函数
from sklearn.model_selection import GridSearchCV

## 定义参数取值范围
learning_rate = [0.1, 0.3, 0.6]
subsample = [0.8, 0.9]
colsample_bytree = [0.6, 0.8]
max_depth = [3,5,8]

parameters = { 'learning_rate': learning_rate,
               'subsample': subsample,
               'colsample_bytree':colsample_bytree,
               'max_depth': max_depth}
model = XGBClassifier(n_estimators = 50)

## 进行网格搜索
clf = GridSearchCV(model, parameters, cv=3,
scoring='accuracy',verbose=1,n_jobs=-1)
clf = clf.fit(x_train, y_train)

## 网格搜索后的最好参数为
clf.best_params_
```

2.4.1 XGBoost的重要参数

1.**eta**[默认0.3]
通过为每一颗树增加权重，提高模型的鲁棒性。
典型值为0.01-0.2。
2.**min_child_weight**[默认1]
决定最小叶子节点样本权重和。
这个参数可以避免过拟合。当它的值较大时，可以避免模型学习到局部的特殊样本。
但是如果这个值过高，则会导致模型拟合不充分。
3.**max_depth**[默认6]
这个值也是用来避免过拟合的。max_depth越大，模型会学到更具体更局部的样本。

典型值：3-10
4.**max_leaf_nodes**
树上最大的节点或叶子的数量。
可以替代max_depth的作用。
这个参数的定义会导致忽略max_depth参数。
5.**gamma**[默认0]
在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma指定了节点分裂所需的最小损失函数下降值。 这个参数的值越大，算法越保守。这个参数的值和损失函数息息相关。
6.**max_delta_step**[默认0]
这参数限制每棵树权重改变的最大步长。如果这个参数的值为0，那就意味着没有约束。如果它被赋予了某个正值，那么它会让这个算法更加保守。
但是当各类别的样本十分不平衡时，它对分类问题是很有帮助的。
7.**subsample**[默认1]
这个参数控制对于每棵树，随机采样的比例。
减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。
典型值：0.5-1
8.**colsample_bytree**[默认1]
用来控制每棵随机采样的列数的占比(每一列是一个特征)。
典型值：0.5-1
9.**colsample_bylevel**[默认1]
用来控制树的每一级的每一次分裂，对列数的采样的占比。
subsample参数和colsample_bytree参数可以起到相同的作用，一般用不到。
10.**lambda**[默认1]
权重的L2正则化项。(和Ridge regression类似)。
这个参数是用来控制XGBoost的正则化部分的。虽然大部分数据科学家很少用到这个参数，但是这个参数在减少过拟合上还是可以挖掘出更多用处的。
11.**alpha**[默认1]
权重的L1正则化项。(和Lasso regression类似)。
可以应用在很高维度的情况下，使得算法的速度更快。
12.**scale_pos_weight**[默认1]
在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。

这个说法很通俗易懂了

XGBoost是基于CART树的集成模型，它的思想是串联多个决策树模型共同进行决策。
那么如何串联呢？XGBoost采用迭代预测误差的方法串联。举个通俗的例子，我们现在需要预测一辆车价值3000元。我们构建决策树1训练后预测为2600元，我们发现有400元的误差，那么决策树2的训练目标为400元，但决策树2的预测结果为350元，还存在50元的误差就交给第三棵树……以此类推，每一颗树用来估计之前所有树的误差，最后所有树预测结果的求和就是最终预测结果！

XGBoost模型可以表示为以下形式，我们约定$f_t(x)$表示前t颗树的和，$h_t(x)$表示第t颗决策树，模型定义如下：

$$f_t(x)=\sum_{t=1}^{T} h_t(x)$$

由于模型递归生成，第t步的模型由第$t-1$步的模型形成，可以写成：

$$f_t(x)=f_{t-1}(x)+h_t(x)$$

每次需要加上的树$h_t(x)$是之前树求和的误差：

$$r_{t,i}=y_i-f_{m-1}(x_i)$$

我们每一步只要拟合一颗输出为$r_{t,i}$的CART树加到$f_{t-1}(x)$就可以了。